grammar.y

```
%{
    #include <stdio.h>
    #include <string.h>
    extern FILE *yyin;
    extern char *yytext;
    extern int yylineno;

    char stringOfProd[1000][3];
    int c = 0;
%}

%token START FINISH DEF NUMBER STRING CHAR ARRAY OF UNDEFINED READ IF STARTIF
FINISHIF ASSIGN
%token WHILE STARTWHILE FINISHWHILE PROC STARTPROC FINISHPROC CALL RETURN
PRINT
%token id constant

%token NEGPOSDIGIT
%token ERRORNUMCONST
%token OP_PLUS OP_MINUS OP_MUL OP_DIV OP_LT OP_LTE OP_EQ OP_NEQ OP_RT OP_RTE
OP_OR OP_AND
%token SEP_SEMICOL SEP_COM SEP_COL SEP_SQBR SEP_SQBREND SEP_RBR SEP_RBREND

%union{
    char varname[26];
    struct attributes
    {
        char varn[20];
    } attrib;
}

%%
program: START cmds FINISH  { char t[3]="0\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
cmds: cmd cmdsconf { char t[3]="1\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
cmdsconf: /*epsilon*/ { char t[3]="2\0";strcpy(stringOfProd[c],t);c+=1;}
        | cmds { char t[3]="3\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
cmd: simplecmd { char t[3]="4\0";strcpy(stringOfProd[c],t);c+=1;}
        | structcmd { char t[3]="5\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
simplecmd: defcmd { char t[3]="6\0";strcpy(stringOfProd[c],t);c+=1;}
        | assigncmd { char t[3]="7\0";strcpy(stringOfProd[c],t);c+=1;}
        | readcmd { char t[3]="8\0";strcpy(stringOfProd[c],t);c+=1;}
        | printcmd { char t[3]="9\0";strcpy(stringOfProd[c],t);c+=1;}
        | returncmd { char t[3]="10\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
defcmd: DEF declist { char t[3]="11\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
declist: declaration declistconf { char
t[3]="12\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
declistconf: /*epsilon*/ { char t[3]="13\0";strcpy(stringOfProd[c],t);c+=1;}
        | SEP_SEMICOL declist { char
t[3]="14\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
declaration: id SEP_COL dtype { char
t[3]="15\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
dtype: primitive { char t[3]="16\0";strcpy(stringOfProd[c],t);c+=1;}
        | arraydecl { char t[3]="17\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
primitive: NUMBER { char t[3]="18\0";strcpy(stringOfProd[c],t);c+=1;}
        | STRING { char t[3]="19\0";strcpy(stringOfProd[c],t);c+=1;}
        | CHAR { char t[3]="20\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
arraydecl: ARRAY SEP_SQBR arraydeclconf { char
t[3]="21\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
arraydeclconf: constant SEP_SQBREND OF primitive { char
t[3]="22\0";strcpy(stringOfProd[c],t);c+=1;}
        | id SEP_SQBREND OF primitive { char
t[3]="23\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
assigncmd: ASSIGN id SEP_COL assigncmdconf { char
t[3]="24\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
assigncmdconf: symbolvalue { char t[3]="25\0";strcpy(stringOfProd[c],t);c+=1;}
        | SEP_RBR expressionstart SEP_RBREND { char
t[3]="26\0";strcpy(stringOfProd[c],t);c+=1;}
        | UNDEFINED { char t[3]="27\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
symbolvalue: id symbolvalueid { char
t[3]="28\0";strcpy(stringOfProd[c],t);c+=1;}
        | constant { char t[3]="29\0";strcpy(stringOfProd[c],t);c+=1;}
        | SEP_SQBR symbolvalueconf { char
t[3]="30\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
symbolvalueid: /*epsilon*/ { char t[3]="31\0";strcpy(stringOfProd[c],t);c+=1;}
        | SEP_SQBR symbolvalueconf { char
t[3]="32\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
symbolvalueconf: id SEP_SQBREND { char
t[3]="33\0";strcpy(stringOfProd[c],t);c+=1;}
```

```
        | constant SEP_SQBREND { char
t[3]="34\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
expressionstart: term expression { char
t[3]="35\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
expression: OP_PLUS term expression { char
t[3]="36\0";strcpy(stringOfProd[c],t);c+=1;}
        | OP_MINUS term expression { char
t[3]="37\0";strcpy(stringOfProd[c],t);c+=1;}
        | /*epsilon*/ { char t[3]="38\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
term: factor muldiv { char t[3]="39\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
muldiv: OP_MUL factor muldiv { char
t[3]="40\0";strcpy(stringOfProd[c],t);c+=1;}
        | OP_DIV factor muldiv { char
t[3]="41\0";strcpy(stringOfProd[c],t);c+=1;}
        | /*epsilon*/ { char t[3]="42\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
factor: SEP_RBR expressionstart SEP_RBREND { char
t[3]="43\0";strcpy(stringOfProd[c],t);c+=1;}
        | symbolvalue { char t[3]="44\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
readcmd: READ id readcmdconf { char
t[3]="45\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
readcmdconf: /*epsilon*/ { char t[3]="46\0";strcpy(stringOfProd[c],t);c+=1;}
        | SEP_SQBR symbolvalueconf { char
t[3]="47\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
printcmd: PRINT SEP_RBR expressionprint SEP_RBREND { char
t[3]="48\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
expressionprint: factorprint expressionprintconf { char
t[3]="49\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
expressionprintconf: /*epsilon*/ { char
t[3]="50\0";strcpy(stringOfProd[c],t);c+=1;}
        | OP_PLUS expressionprint { char
t[3]="51\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
factorprint: id { char t[3]="52\0";strcpy(stringOfProd[c],t);c+=1;}
        | constant { char t[3]="53\0";strcpy(stringOfProd[c],t);c+=1;}
        | callstmt { char t[3]="54\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
returncmd: RETURN returncmdconf { char
t[3]="55\0";strcpy(stringOfProd[c],t);c+=1;}
```

```
        ;
returncmdconf: expressionstart { char
t[3]="56\0";strcpy(stringOfProd[c],t);c+=1;}
        | callstmt { char t[3]="57\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
structcmd: ifstmt { char t[3]="58\0";strcpy(stringOfProd[c],t);c+=1;}
        | whilestmt { char t[3]="59\0";strcpy(stringOfProd[c],t);c+=1;}
        | procstmt { char t[3]="60\0";strcpy(stringOfProd[c],t);c+=1;}
        | callstmt { char t[3]="61\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
ifstmt: IF condition STARTIF cmds FINISHIF { char
t[3]="62\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
condition: basiccondition conditionconf { char
t[3]="63\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
conditionconf: /*epsilon*/ { char t[3]="64\0";strcpy(stringOfProd[c],t);c+=1;}
        | logicaloperator condition { char
t[3]="65\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
basiccondition: symbolvalue comparisonoperator basicconditionconf { char
t[3]="66\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
basicconditionconf: symbolvalue { char
t[3]="67\0";strcpy(stringOfProd[c],t);c+=1;}
        | UNDEFINED { char t[3]="68\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
comparisonoperator: OP_LT { char t[3]="69\0";strcpy(stringOfProd[c],t);c+=1;}
        | OP_RT { char t[3]="70\0";strcpy(stringOfProd[c],t);c+=1;}
        | OP_LTE { char t[3]="71\0";strcpy(stringOfProd[c],t);c+=1;}
        | OP_RTE { char t[3]="72\0";strcpy(stringOfProd[c],t);c+=1;}
        | OP_EQ { char t[3]="73\0";strcpy(stringOfProd[c],t);c+=1;}
        | OP_NEQ { char t[3]="74\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
logicaloperator: OP_AND { char t[3]="75\0";strcpy(stringOfProd[c],t);c+=1;}
        | OP_OR { char t[3]="76\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
whilestmt: WHILE condition STARTWHILE cmds FINISHWHILE { char
t[3]="77\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
procstmt: PROC id SEP_RBR procstmtconf { char
t[3]="78\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
procstmtconf: SEP_RBREND STARTPROC cmds FINISHPROC { char
t[3]="79\0";strcpy(stringOfProd[c],t);c+=1;}
        | declist SEP_RBREND STARTPROC cmds FINISHPROC { char
t[3]="80\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
```

```
callstmt: CALL id SEP_RBR paramslist SEP_RBREND { char
t[3]="81\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
paramslist: expressionstart paramslistconf { char
t[3]="82\0";strcpy(stringOfProd[c],t);c+=1;}
        | /*epsilon*/ { char t[3]="83\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
paramslistconf: /*epsilon*/ { char
t[3]="84\0";strcpy(stringOfProd[c],t);c+=1;}
        | SEP_COM paramslist { char
t[3]="85\0";strcpy(stringOfProd[c],t);c+=1;}
        ;
%%

int main(int argc, char **argv)
{
    if (argc == 2) {
        yyin = fopen(argv[1], "r");
        yyparse();
    }
    else{
        printf("No input file given!\n");
    }
    if(0==yyparse()) printf("Result yyparse OK");
}

void reverseString(){
    char aux[3];
    for(int i=0;i<c / 2;i++){
        strcpy(aux, stringOfProd[i]);
        strcpy(stringOfProd[i], stringOfProd[c-i-1]);
        strcpy(stringOfProd[c-i-1], aux);
    }
}

int yyerror(char *s)
{
    printf("String of productions: \n");
    reverseString(stringOfProd);
    for(int i=0;i<c;i++){
        printf("%s ", stringOfProd[i]);
    }
    printf("\n");
    printf("Error on line #%d\n", yylineno);
    printf("Unexpected token: '%s'\n", yytext);
    return 0;
}
```

ex:

START

PRINT(10)

FINISH

productions:

0 program -> START cmds FINISH

1 cmds -> cmd cmdsconf

2 cmdsconf -> Epsilon

4 cmd -> simplecmd

9 simplecmd -> printcmd

48 printcmd -> PRINT [expressionprint]

49 expressionprint -> factorprint expressionprintf

50 expressionprintconf -> Epsilon

53 factorprint -> constant